

## Anotações:

SLL = Shift Left Logical  
SRL = Shift Right Logical

## Atividade 1

Faça um programa em assembly que que leia x e y do teclado, e realize a equação (utilizando deslocamentos):

$z = ((12 * x) + (66 * y)) * 4;$

Compare o comportamento com o uso das instruções **mul** e **div** (quando disponíveis).

The screenshot shows the RISC-V Simulator interface. On the left, the assembly code is displayed with line numbers 9 to 25. The code includes instructions for adding zero to zero, moving values between registers, and performing shifts. On the right, the Register window shows the current values of registers zero through t6. The registers are organized in two columns: zero, ra, sp, tp, t1, s0/fp, a0, a2, a4, a6, s2, s4, s6, s8, s10, t3, t5 on the left; and ra, gp, t0, t2, t4, s1, a1, a3, a5, a7, s3, s5, s7, s9, s11, t4, t6 on the right. The values are shown in hexadecimal.

Register	Value
zero [0]	0
ra [1]	12
sp [2]	1536
gp [3]	0
tp [4]	0
t0 [5]	4
t1 [6]	4
t2 [7]	120
s0/fp [8]	0
s1 [9]	0
a0 [10]	20
a1 [11]	10
a2 [12]	20
a3 [13]	80
a4 [14]	40
a5 [15]	1280
a6 [16]	40
a7 [17]	5760
s2 [18]	0
s3 [19]	0
s4 [20]	0
s5 [21]	0
s6 [22]	0
s7 [23]	0
s8 [24]	0
s9 [25]	0
s10 [26]	0
s11 [27]	0
t3 [28]	1320
t4 [29]	1440
t5 [30]	0
t6 [31]	0

## Resposta:

- Qual instrução é mais simples de executar no nível de hardware?  
Os deslocadores de bits: SRL e SLL.
- O que acontece se o número for negativo e você usar `srl` (deslocamento lógico à direita)?  
Ele vira um número positivo, e muda o valor. Por exemplo, -8 é 11111000, ao usar o `srl` ele vira 01111100.

sar (Shift Arithmetic Right).

## Atividade 2

Faça um programa que leia um número do teclado e imprima a letra I se o número for ímpar ou a letra P se o número for par. Busque os códigos da letra I e P na tabela ASCII para poder imprimir o caractere corretamente.

RISC-V Assembly

```

6      addi    zero,zero,0
7      auipc   ra,0x0
8      jalr    ra,0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0

main:
11     addi    t0, zero, 4
12     ecall

13     mv      a1, a0

14     andi    t1, a1, 1

15     beqz    t1, eh_par

eh_impar:
16     addi    a0, zero, 73
17     addi    t0, zero, 2
18     ecall
19     j      fim_programa

eh_par:
20     addi    a2, zero, 80
21     addi    t0, zero, 2
22     ecall

fim_programa:
23     j      fim
24     addi    zero,zero,0
25     addi    zero,zero,0
26     addi    zero,zero,0
27     addi    zero,zero,0
28     auipc   ra,0x0
29     jalr    ra,0(ra)
30     addi    zero,zero,0
31     addi    zero,zero,0
32     addi    zero,zero,0
33     addi    zero,zero,0

```

Registers

Register	Value	Register	Value		
zero	[0]	0	ra	[1]	12
sp	[2]	1536	gp	[3]	0
tp	[4]	0	t0	[5]	2
t1	[6]	1	t2	[7]	0
s0/fp	[8]	0	s1	[9]	0
a0	[10]	73	a1	[11]	25
a2	[12]	0	a3	[13]	0
a4	[14]	0	a5	[15]	0
a6	[16]	0	a7	[17]	0
s2	[18]	0	s3	[19]	0
s4	[20]	0	s5	[21]	0
s6	[22]	0	s7	[23]	0
s8	[24]	0	s9	[25]	0
s10	[26]	0	s11	[27]	0
t3	[28]	0	t4	[29]	0
t5	[30]	0	t6	[31]	0

Instruction breakdown

31	12	11	7
imm[20 10:1 11 19:12]		rd	
00000000000000000000		00000	

Console

```

***** Parser Output *****
Parsing successful!
15
***** Parser Output *****
Parsing successful!
25
I

```

### Atividade 3

Você consegue fazer outra versão do programa que detecte se um número é múltiplo de 4? Nesse caso, imprima S para sim e N para não.

**Dica:** A instrução OR faz uma operação OU lógica bit a bit, para todos os bits do número. Então, um OR entre os números 6 (0110) e 5 (0101) tem como resposta o número 7 (0111) pois todos os bits com valor 1 em, ao menos, um dos números foram mantidos como 1 no resultado final.

RISC-V Assembly

⬆

▶

⏮

⏭

```
main:
11      addi t0, zero, 4
12      ecall

13      addi a1, zero, 3
14      or a2, a0, a1
15      addi a3, a0, 3

16      bne a2, a3, nao_multiplo

eh_multiplo:
17      addi a0, zero, 83
18      addi t0, zero, 2
19      ecall
20      j fim_programa

nao_multiplo:
21      addi a0, zero, 78
22      addi t0, zero, 2
23      ecall

fim_programa:
24      j fim
25      addi zero, zero, 0
26      addi zero, zero, 0
27      addi zero, zero, 0
28      addi zero, zero, 0
29      auipc ra, 0x0
30      jalr ra, 0(ra)
31      addi zero, zero, 0
32      addi zero, zero, 0
33      addi zero, zero, 0
34      addi zero, zero, 0
```

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 12
sp	[2] 1536	gp	[3] 0
tp	[4] 0	t0	[5] 2
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 78	a1	[11] 3
a2	[12] 7	a3	[13] 8
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

```
Parsing successful!
5
N4
N***** Parser Output *****
Parsing successful!
4
S5
N
```

Instruction breakdown

31	12	11
imm[20:10:11:19:12]		rd
00000000000000000000		00000

## Atividade 4

Faça um programa que leia múltiplos números do teclado. Seu programa deve parar quando for digitado o valor 0. Ao final do programa, ele deve imprimir o resultado da soma de todos os ímpares subtraindo da soma de todos os pares. Resultado = Soma(ímpares) - Soma(pares).

RISC-V Assembly

⬆️

▶️

⏮️

⏭️

```
8      jalr    ra,0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0

main:
11     add     s2, zero, zero
12     add     s1, zero, zero

loop:
13     addi    t0, zero, 4
14     ecall

15     beqz    a0, fim_loop

16     andi    t1, a0, 1
17     beqz    t1, par

impar:
18     add     s1, s1, a0
19     j       loop

par:
20     add     s2, s2, a0
21     j       loop

fim_loop:
22     sub     a0, s1, s2
23     addi    t0, zero, 1
24     ecall

fim_programa:
25     j       fim

# 5+5 = 10
# 2+2 = 4
# 10 - 4 = 6
26     addi    zero,zero,0
27     addi    zero,zero,0
28     addi    zero,zero,0
29     addi    zero,zero,0
30     auipc   ra,0x0
31     jalr    ra,0(ra)
32     addi    zero,zero,0
33     addi    zero,zero,0
34     addi    zero,zero,0
```

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 12
sp	[2] 1536	gp	[3] 0
tp	[4] 0	t0	[5] 1
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 10
a0	[10] 6	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 4	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

\*\*\*\*\* Parser Output \*\*\*\*\*

Parsing successful!

5

5

2

2

0

6

Instruction breakdown

31

imm[20:10:1|11:19:12]

00000000000000000000

## Atividade 5

Faça um programa que leia dois números do teclado: o segredo e o número a codificar. O programa deve imprimir o número codificado. Para isso, utilize a instrução XOR. Note que esse programa servirá tanto para codificar quanto para decodificar o número.

**⚠️ Atenção:** Apesar de ser uma forma de codificar um número, essa técnica não é segura. Ela é apenas uma forma de você aprender a utilizar a instrução XOR.

RISC-V Assembly

⬆️

▶️

⏮️

⏪️

5

addi

zero,zero,0

6

addi

zero,zero,0

7

auipc

ra,0x0

8

jalr

ra,0(ra)

9

addi

zero,zero,0

10

addi

zero,zero,0

main:

11

addi

t0, zero, 4

12

ecall

13

mv

a1, a0

14

addi

t1, zero, 4

15

ecall

16

xor

a0, a0, a1

17

addi

t0, zero, 1

18

ecall

19

addi

zero,zero,0

20

addi

zero,zero,0

21

addi

zero,zero,0

22

addi

zero,zero,0

23

auipc

ra,0x0

24

jalr

ra,0(ra)

25

addi

zero,zero,0

26

addi

zero,zero,0

27

addi

zero,zero,0

Registers

Register		Value	Register		Value
zero	[0]	0	ra	[1]	12
sp	[2]	1536	gp	[3]	0
tp	[4]	0	t0	[5]	1
t1	[6]	4	t2	[7]	0
s0/fp	[8]	0	s1	[9]	0
a0	[10]	182	a1	[11]	210
a2	[12]	0	a3	[13]	0
a4	[14]	0	a5	[15]	0
a6	[16]	0	a7	[17]	0
s2	[18]	0	s3	[19]	0
s4	[20]	0	s5	[21]	0
s6	[22]	0	s7	[23]	0
s8	[24]	0	s9	[25]	0
s10	[26]	0	s11	[27]	0
t3	[28]	0	t4	[29]	0
t5	[30]	0	t6	[31]	0

Instruction breakdown

31	20	19	15	14	12	11
imm		rs1		funct3		rd
000000000000		00000		000		000

Console

\*\*\*\*\* Parser Output \*\*\*\*\*

Parsing successful!

5

5

2

2

0

6

\*\*\*\*\* Parser Output \*\*\*\*\*

Parsing successful!

5

3

6

210

100

182

210

100

182

## Atividade 6

Faça um programa que leia o segredo do teclado e, depois, leia vários números do teclado imprimindo cada um codificado em sequência. O programa deve parar quando for digitado o valor 0.

RISC-V Assembly

↑

▶

⏮

⏭

```
0      addi    zero,zero,0
kernel:
main:
11     addi    t0, zero, 4
12     ecall
13
14     mv      a1, a0
loop:
14     addi    t0, zero, 4
15     ecall
16
17     beqz    a0, fim_programa
18
19     xor      a0, a0, a1
20
21     addi    t0, zero, 1
22     ecall
23
24     j       loop
fim_programa:
21     addi    zero,zero,0
22     addi    zero,zero,0
23     addi    zero,zero,0
24     addi    zero,zero,0
25     auipc   ra,0x0
26     jalr    ra,0(ra)
27     addi    zero,zero,0
28     addi    zero,zero,0
```

Console

20

30

30

\*\*\*\*\* Parser Output \*\*\*\*\*

Parsing successful!

10

20

30

50

56

11

1

2

8

0

\*\*\*\*\* Parser Output \*\*\*\*\*

Parsing successful!

10

5

15

0

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 108
sp	[2] 1536	gp	[3] 0
tp	[4] 0	t0	[5] 4
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 10
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Instruction breakdown

31	12	11
imm[31:12]		rd
00000000000000000000		00001

## Atividade 7

Faça um programa que leia um número do teclado e imprima o número em binário. Para isso, utilize as instruções AND e de deslocamento de bits. Como os números do seu processador são de 32 bits, você deve ter um laço for no seu código com 32 interações.

[illegible]